Date=21/08/2020
Lecture By=Shubham Joshi
Subject ⇒Dynamic Programing

IN PREVIOUS LECTURE **(QUICK RECAP) Date-20/08/2020**	**In Today's Lecture (Overview)**
⇒ Insertion of heap Question=2(From leetcode) MCQs Questions For Self practice // CC For The Day	⇒ Dynamic programing in python ⇒ Program to Print fibonacci series using dynamic programing ⇒ MCQs ⇒ Questions for self practice // CC and Assignment for the day

⇒ Dynamic programing in python

Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems
But unlike, divide and conquer, these sub-problems are not solved independently.
Rather, results of these smaller subproblems are remembered and used for similar or overlapping sub-problems.
Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be reused
Mostly, these algorithms are used for optimization.

So we can say that −

- The problem should be able to be divided into smaller overlapping sub-problem.
- An optimum solution can be achieved by using an optimum solution of smaller sub-problems.
- Dynamic algorithms use Memoization.

Dynamic programming can be used in both top-down and bottom-up manner. And of course, most of the time, referring to the previous solution output is cheaper than

recomputing in terms of CPU cycles.

Steps to Designing a Dynamic Programming Algorithm

1. Characterize optimal substructure

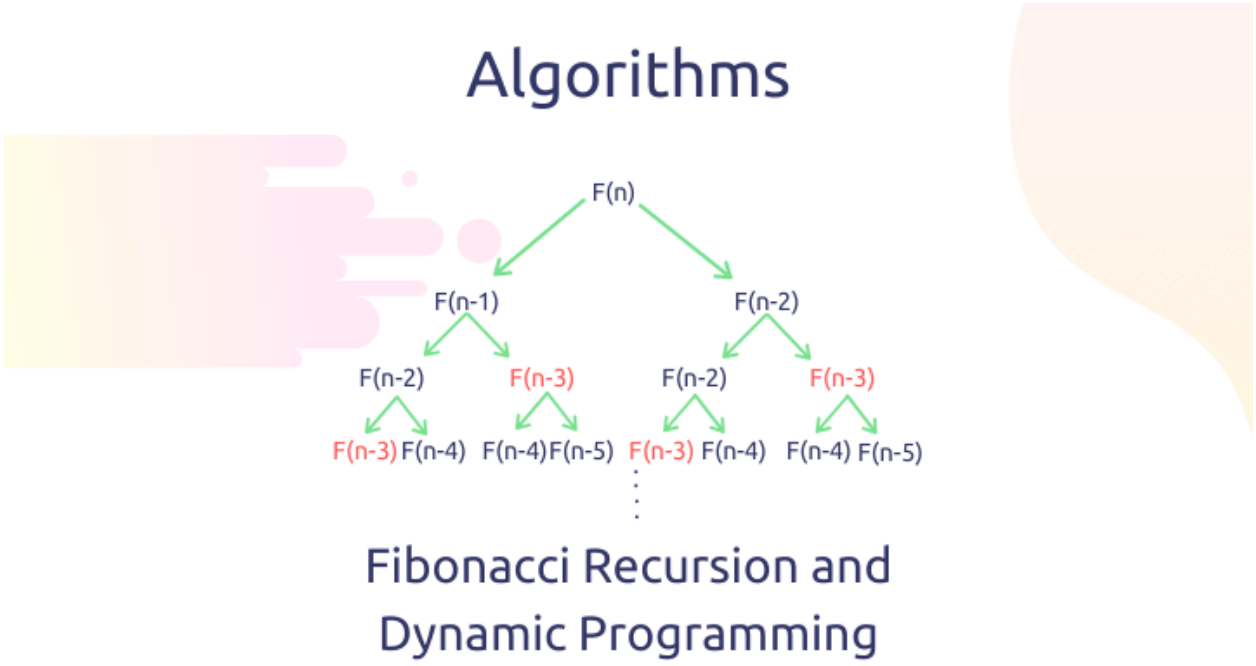
2. Recursively define the value of an optimal solution

3. Compute the value bottom up

4. (if needed) Construct an optimal solution

	Tabulation	Memoization
State	State Transition relation is difficult to think	State transition relation is easy to think
Code	Code gets complicated when lot of conditions are required	Code is easy and less complicated
Speed	Fast, as we directly access previous states from the table	Slow due to lot of recursive calls and return statements
Subproblem solving	If all subproblems must be solved at least once, a bottom-up dynamic-programming algorithm usually outperforms a top-down memoized algorithm by a constant factor	If some subproblems in the subproblem space need not be solved at all, the memoized solution has the advantage of solving only those subproblems that are definitely required
Table Entries	In Tabulated version, starting from the first entry, all entries are filled one by one	Unlike the Tabulated version, all entries of the lookup table are not necessarily filled in Memoized version. The table is filled on demand.

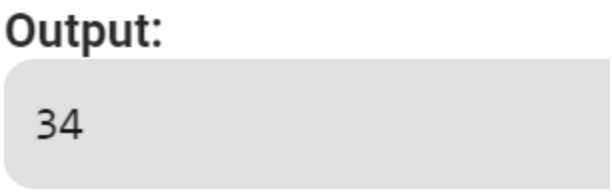
⇒ Program to Print fibonacci series using dynamic programing




```
# Fibonacci Series using Dynamic Programming
def fibonacci(n):

    # Taking 1st two fibonacci numbers as 0 and 1
    FibArray = [0, 1]

    while len(FibArray) < n + 1:
        FibArray.append(0)

    if n <= 1:
        return n
    else:
        if FibArray[n - 1] == 0:
            FibArray[n - 1] = fibonacci(n - 1)

        if FibArray[n - 2] == 0:
            FibArray[n - 2] = fibonacci(n - 2)
```

```
    FibArray[n] = FibArray[n - 2] + FibArray[n - 1]
    return FibArray[n]

print(fibonacci(9))
```

Output



⇒ MCQs

1.**What of this is a characteristic of dp ?**

A=overlapping subproblems

B=it is recursion

C=it uses queues data structure

2.What is the thing based on which we save our previous computation in dp ?

A=function calls

B=states which are changing in recursion

3.What is true about dp ?

A=It reduces the time complexity of recursive solution where same state is getting called again and again

B=it is of 3 types

C=it has no effect on time complexity of program

4.In which of these we build solutions from bottom up ?

A=memoization

B=tabular

⇒ Questions for self practice // CC and Assignment for the day

1.https://leetcode.com/problems/climbing-stairs/

Try to make recursive functions for these two questions

2.https://leetcode.com/problems/house-robber/

3.https://leetcode.com/problems/coin-change/